## Useful sources

* Lots of sample code snippers can be found in the [ScriptRepository](https://www.slicer.org/wiki/Documentation/Nightly/ScriptRepository)
* There are a great number of Slicer usage and coding related questions answered on the [Slicer discourse](https://discourse.slicer.org/)
* [Slicer developer guide](https://slicer.readthedocs.io/en/latest/developer_guide/index.html)
* Always browse the Slicer source code for example usages when in doubt  
  (e.g. look up the usage of a certain class with grep or Alt+F7 in Total Commander or Double Commander)
* [Qt documentation](https://doc.qt.io/qt-5/)
* [VTK documentation](https://vtk.org/documentation/) and [Class list](https://vtk.org/doc/nightly/html/classes.html)

## Build

It is important to note that the FirstTask application is not a Slicer module that *runs on* regular Slicer, but its own application that *uses* Slicer. So you’ll need to build the FirstTask code and not the Slicer code. Please see [Slicer build instructions](https://slicer.readthedocs.io/en/latest/developer_guide/build_instructions/index.html), and the build scripts are available in the Scripts folder in the FirstTask repository. It is very important to use Qt 5.15.1, so double check that it is the one used.

On Windows make sure you use as short a path as possible for the superbuild directories, otherwise build may fail. For example, in this case e:\f\fR

## GitHub workflow

1. Create a ticket about the task you are doing
2. Make yourself assigned
3. Fetch origin, switch to *master* branch, rebase *origin/master*
4. Create topic branch
   1. For ticket with number N the branch name is ‘N-description-of-task’
   2. Switch to topic branch
5. Implement change
6. When done, push
7. Create pull request to the upstream (origin) master
   1. Add *cpinter* as Reviewer

## Developer mode

Turn on Developer mode to see the Settings section on the bottom of the slicelet panel, in order to be able to show/hide the Python console and switch to full Slicer application.

You will need to edit the application settings file.

* Linux: ~/.config/Ebatinca, S.L/FirstTask.ini
* Windows: c:\Users\[UserName]\AppData\Roaming\Ebatinca, S.L\FirstTask.ini

Open the file in a text editor, and change the *DeveloperMode* setting in the *[Developer]* section to true:

[Developer]  
DeveloperMode=true

## Python development

### Python syntax errors

If instead of the custom application a general Slicer window opens, it means there is a syntax error in your code or an issue that prevents proper initialization of the FirstTask module. You will need to open the Error log (in View menu, or press Ctrl+0), and find the first message starting with “*Traceback*”. See the error, fix it, and restart the application.

### Using the Python interactor for development

If you are not confident about a certain command or loop (e.g. what effect setting a property exactly has, how to traverse a certain container etc.), then it is more efficient to try out in the Python interactor and then copy-paste the verified code to the proper Python file than editing the file and starting the application many times.

This is the reason the *slicer.firstTaskWidget* variable is set up. After loading the test data you need to develop the code you want, you can access any object through this variable. For example:

slicer.firstTaskWidget.mainMenuDockWidget

### Copy only the Python source files to speed up development

Test application after each code change that only affect .py or .ui files without running the superbuild (which is really slow):

Copy the source files to their proper places. This is a Windows batch script but the same applies to Linux with the appropriate syntax:

copy /Y e:\f\FirstTaskApp\Modules\FirstTask\FirstTask.py e:\f\fR\Slicer-build\lib\FirstTask-4.11\qt-scripted-modules  
copy /Y e:\f\FirstTaskApp\Modules\FirstTask\Resources\UI\\*.ui e:\f\fR\Slicer-build\lib\FirstTask-4.11\qt-scripted-modules\Resources\UI

(e:\f\fR is the Release superbuild directory)

## Using Qt Designer properly

Qt Designer **must** be started with the Slicer environment so that the CTK and Slicer widgets are properly handled. It can be started with the main custom app launcher like this:

./[DEVEL\_DIR]/Slicer-build/FirstTask --designer

If this command starts the FirstTask application and not the Qt Designer (may happen on Linux or Mac), contact Csaba who can give you a release that allows opening Qt Designer with the proper environment.

Please note that if compilation fails due to unable to write a binary file when the designer is open like this, it may need to be closed before the build succeeds

## Using the startup script .slicerrc.py

The contents of the *.slicerrc.py* file are executed at each Slicer or Slicer-based application startup, so you can enter code there you want to automatically be run.

### Use test data for development

It’s easier to develop if you have the necessary data loaded at startup. Ask Csaba for test data for the particular use case you are working on.

To load your test data you will need to switch to the DICOM database that you use for this development, then load the data by UID (first of course you have to have a database and import the data in the regular Slicer DICOM browser).

Switch to database (*r’’* means don’t unescape characters, useful on Windows):

DICOMUtils.openDatabase(r'd:\data\SlicerData\\_DICOM\_Database')

Load data:

DICOMUtils.loadPatientByName('liver\_04^patient')

It is possible to load entire scenes if you use a scene for testing:

slicer.util.loadScene(r'd:\data\FirstTask\Scene.mrb')

If any volume data serves for testing, then you can use a dataset from the Sample Data module:

import SampleData  
mrNode = SampleData.SampleDataLogic().downloadMRHead()

### Other useful snippets to put in slicerrc

I tend to hide the annoying and large Slicer logo from the panel:

if hasattr(slicer.util, 'setApplicationLogoVisible'):  
 slicer.util.setApplicationLogoVisible(False)

Or make the Python auto-complete menu longer:

if hasattr(slicer.app.pythonConsole(), 'maxVisibleCompleterItems'):  
 slicer.app.pythonConsole().maxVisibleCompleterItems = 32